## AMENDMENTS TO THE CLAIMS

This listing of claims will replace all prior versions, and listings, of claims in the application:

**Listing of Claims:**

1. (Currently Amended) A method for loading classes into memory, comprising:

loading class definitions into memory;

wherein the class definitions contain metadata for classes that are currently being loaded into memory, as well as metadata for classes that are already loaded into memory; and

after the class definitions are loaded into memory, loading method code for the classes into memory;

wherein loading the method code into memory involves transforming the method code, wherein transforming the method code involves using the class definitions to resolve linkages in the method code so that the method code is ready for execution in memory.

2. (Original) The method of claim 1, wherein the class definitions are loaded into volatile memory and the method code is loaded into non-volatile memory.

3. (Original) The method of claim 2, wherein after the method code is loaded into non-volatile memory, the method further comprises using the class definitions to create class data structures for the classes in non-volatile memory.

4. (Original) The method of claim 3, wherein prior to creating the class data structures in non-volatile memory, the method further comprises creating one or more jump tables in non-volatile memory, wherein the jump tables specify the locations of methods.

5. (Original) The method of claim 3, wherein after the class data structures are created in non-volatile memory, the method further comprises deleting the class definitions from volatile memory.

6. (Original) The method of claim 2, wherein resolving linkages in the method code involves quickening the method code by resolving symbolic references into either offset-based references or pointer-based references.

7. (Original) The method of claim 1,

wherein the classes are loaded from a suite file containing the classes; and

wherein the suite file is organized so that the class definitions for all of the classes in the suite file precede the method code for the classes, thereby facilitating loading the class definitions prior to loading the method code.

8. (Original) The method of claim 2, wherein during the loading of the method code into non-volatile memory, the method code is verified to ensure that the method code is correct with regards to type safety.

9. (Original) The method of claim 2, wherein after the method code is loaded into non-volatile memory, the method code is verified to ensure that branch targets within the method code are valid.

10. (Original) The method of claim 2, wherein the class definitions include:

real class definitions for classes that are currently being loaded into non-volatile memory; and

proxy class definitions for classes that were previously loaded into non-volatile memory.

11. (Original) The method of claim 2,

wherein the volatile memory is Random Access Memory (RAM); and

wherein the non-volatile memory is Electrically-Erasable Read-Only Memory (EEPROM).

12. (Currently Amended) A computer-readable storage medium storing instructions that when executed by a computer cause the computer to perform a method for loading classes into memory, the method comprising:

loading class definitions into memory;

wherein the class definitions contain metadata for classes that are currently being loaded into memory, as well as metadata for classes that are already loaded into memory; and

after the class definitions are loaded into memory, loading method code for the classes into memory;

wherein loading the method code into memory involves transforming the method code, wherein transforming the method code involves using the class definitions to resolve linkages in the method code so that the method code is ready for execution in memory.

13. (Original) The computer-readable storage medium of claim 12, wherein the class definitions are loaded into volatile memory and the method code is loaded into non-volatile memory.

14. (Original) The computer-readable storage medium of claim 13, wherein after the method code is loaded into non-volatile memory, the method further comprises using the class definitions to create class data structures for the classes in non-volatile memory.

15. (Original) The computer-readable storage medium of claim 14, wherein prior to creating the class data structures in non-volatile memory, the method further comprises creating one or more jump tables in non-volatile memory, wherein the jump tables specify the locations of methods.

16. (Original) The computer-readable storage medium of claim 14, wherein after the class data structures are created in non-volatile memory, the method further comprises deleting the class definitions from volatile memory.

17. (Original) The computer-readable storage medium of claim 13, wherein resolving linkages in the method code involves quickening the method code by resolving symbolic references into either offset-based references or pointer-based references.

18. (Original) The computer-readable storage medium of claim 12,

wherein the classes are loaded from a suite file containing the classes; and

wherein the suite file is organized so that the class definitions for all of the classes in the suite file precede the method code for the classes, thereby facilitating loading the class definitions prior to loading the method code.

19. (Original) The computer-readable storage medium of claim 13, wherein during the loading of the method code into non-volatile memory, the method code is verified to ensure that the method code is correct with regards to type safety.

20. (Original) The computer-readable storage medium of claim 13, wherein after the method code is loaded into non-volatile memory, the method code is verified to ensure that branch targets within the method code are valid.

21. (Original) The computer-readable storage medium of claim 13, wherein the class definitions include:

real class definitions for classes that are currently being loaded into non-volatile memory; and

proxy class definitions for classes that were previously loaded into non-volatile memory.

22. (Original) The computer-readable storage medium of claim 13,

wherein the volatile memory is Random Access Memory (RAM); and

wherein the non-volatile memory is Electrically-Erasable Read-Only Memory (EEPROM).

23. (Currently Amended) An apparatus that loads classes into memory, comprising:

a loading mechanism;

wherein the loading mechanism is configured to load class definitions into memory;

wherein the class definitions contain metadata for classes that are currently being loaded into memory, as well as metadata for classes that are already loaded into memory; and

wherein after the class definitions are loaded into memory, the loading mechanism is configured to load method code for the classes into memory;

wherein loading the method code into memory involves transforming the method code, wherein transforming the method code involves using the class definitions to resolve linkages in the method code so that the method code is ready for execution in memory.

24. (Original) The apparatus of claim 23, wherein the class definitions are loaded into volatile memory and the method code is loaded into non-volatile memory.

25. (Original) The apparatus of claim 24, wherein after the method code is loaded into non-volatile memory, the loading mechanism is configured to use the class definitions to create class data structures for the classes in non-volatile memory.

26. (Original) The apparatus of claim 25, wherein prior to creating the class data structures in non-volatile memory, the loading mechanism is configured to create one or more jump tables in non-volatile memory, wherein the jump tables specify the locations of methods.

27. (Original) The apparatus of claim 25, wherein after the class data structures are created in non-volatile memory the loading mechanism is configured to delete the class definitions from volatile memory.

28. (Original) The apparatus of claim 24, wherein the loading mechanism is configured to resolve linkages in the method code by quickening the method code to resolve symbolic references into either offset-based references or pointer-based references.

29. (Original) The apparatus of claim 23,

wherein the classes are loaded from a suite file containing the classes; and

wherein the suite file is organized so that the class definitions for all of the classes in the suite file precede the method code for the classes, thereby facilitating loading the class definitions prior to loading the method code.

30. (Original) The apparatus of claim 24, wherein during loading of the method code into non-volatile memory, the loading mechanism is configured to verify the method code to ensure that the method code is correct with regards to type safety.

31. (Original) The apparatus of claim 24, wherein after the method code is loaded into non-volatile memory, the loading mechanism is configured to verify that branch targets within the method code are valid.

32. (Original) The apparatus of claim 24, wherein the class definitions include:

real class definitions for classes that are currently being loaded into non-volatile memory; and

proxy class definitions for classes that were previously loaded into non-volatile memory.

33. (Original) The apparatus of claim 24,

wherein the volatile memory is Random Access Memory (RAM); and wherein the non-volatile memory is Electrically-Erasable Read-Only Memory (EEPROM).

34. (Currently Amended) A computing device configured to load classes into non-volatile memory, comprising:

a computing engine;

a volatile memory;

a non-volatile memory;

a loading mechanism;

wherein the loading mechanism is configured to load class definitions into the volatile memory;

wherein the class definitions contain metadata for classes that are currently being loaded into non-volatile memory, as well as metadata for classes that are already loaded into non-volatile memory; and

wherein after the class definitions are loaded into volatile memory, the loading mechanism is configured to load method code for the classes into the non-volatile memory;

wherein loading the method code into the non-volatile memory involves transforming the method code, wherein transforming the method code involves using the class definitions to resolve linkages in the method code so that the method code is ready for execution in the non-volatile memory.